AC 2007-1274: EDUCATIONAL DISCRETE TIME SIGNAL PROCESSING TOOLKIT

Jonathan Hill, University of Hartford

Dr. Jonathan Hill is an assistant professor in the College of Engineering, Technology, and Architecture (CETA) at the University of Hartford, Connecticut (USA). Ph.D. and M.S. from Worcester Polytechnic Institute (WPI) and B.S. from Northeastern University. Previously an applications engineer with the Networks and Communications division of Digital Corporation. His interests involve embedded microprocessor based systems.

Hisham Alnajjar, University of Hartford

Dr. Hisham Alnajjar is the chair for the Electrical and Computer Engineering Dept. at the University of Hartford, Connecticut (USA), where he is also the assistant dean of the College of Engineering, Technology, and Architecture (CETA). Ph.D. from Vandebilt University, M.S. from Ohio University. His research interests include sensor array processing, digital signal processing, and power systems.

Saeid Moslehpour, University of Hartford

Dr. Saeid Moslehpour is an assistant professor in the electrical and computer engineering department in the College of Engineering, Technology, and Architecture (CETA) at the University of Hartford. PhD from Iowa State University and B.S. M.S. and Ed.Sp. degrees from Central Missouri State University. His areas of interest are logic design, CPLDs, FPGAs and distance learning.

Educational Discrete Time Signal Processing Toolkit

Abstract

The field programmable gate array (FPGA) provides new ways for students to investigate discrete time signal processing principles. In teaching signal processing, we find that students typically lack an intuitive feel for discrete time signals. Basic topics such as sampling have subtleties that plague students. To be useful in helping students to develop such an intuition, it is important that the tools be simple and that no detail be hidden. Unlike existing software, all details must be visible in a simple yet transparent fashion. Second, students need useful tools for developing their own projects.

We are developing the discrete time signal processing toolkit for a digital signal processing (DSP) course, to be useful to students learning DSP principles as well as to advanced students working on their own projects. As an introductory tool, the toolkit will allow a deductive approach where students investigate existing systems. Advanced students ready for a more inductive approach can use the toolkit in their own projects by drawing schematics or modifying example VHDL modules. Students are not expected to write code using a hardware description language, but the underlying code is always available for inspection.

The toolkit is multipurpose that along with course materials provides several methods to process signals. First off, the toolkit demonstrates signal conversion, sampling, aliasing, and the importance of a reconstruction filter. Fixed point numbers are discussed and a simple IIR filter is presented. A simple structure used to perform convolution is used as a building block to construct FIR filters. The toolkit can optionally use a soft core digital signal processor. Details of the toolkit are available on the project website[1].

Introduction

The discrete time signal processing toolkit is useful to students learning digital signal processing principles as well as advanced students working on their own projects. In teaching signal processing, we find that students typically lack an intuitive feel for discrete time signals. Basic topics such as sampling have subtleties that plague students. The discrete time signal processing toolkit is being developed for a digital signal processing course, to be useful in helping students to develop such an intuition, it is important that the tools be simple and that all details be visible in a simple yet transparent fashion.

Details of the toolkit are available on the project website[1]. The toolkit is currently based on the Xilinx[4] Spartan-3 FPGA Starter Board, which is an off-the shelf development board from Digilent, Inc[2]. The development board is used with a custom acquisition card that we designed. Analog to digital conversion is performed with the Analog Devices[3] AD7819 chip which produces 8 bit samples at rates up to 200k samples per second. Digital to analog conversion is performed using a conventional R-2R ladder network. The acquisition card has a solderless bread board area for students to construct analog input and output filters.

Several options exist for configuring FPGA resources. The Xilinx ISE/Webpack[4] tools are freely available for students to download and use. However the code is fairly generic and should be vendor independent. In some cases as in the preliminary experience the circuit is described conveniently in VHDL. In the remainder of the experiences a combination of VHDL and schematic capture tools are used. As an introductory tool, the toolkit allows a deductive approach where students investigate existing systems. Advanced students ready for a more inductive approach can use the toolkit in their own projects by drawing schematics or modifying example VHDL modules. Students are not expected to write code using a hardware description language, but the underlying code is always available for inspection.

Preliminary Experience

The first experience students have with the toolkit involves basic practical concerns that students must become familiar with. The first topics to address are how to convert from analog to digital and digital back to analog. Figure 1 is the conceptual diagram of a test circuit that students use along with a voltmeter. A potentiometer is sampled by an analog to digital converter and the result is displayed in hexadecimal. Switches provide a digital value which is converted to the analog voltage Vo. By means of push buttons, the LED bar graph displays either the converted input or switch values. The ADC is sampled at 10Hz and the seven-segment display is time multiplexed, transitioning between digits at a 1 kHz rate.

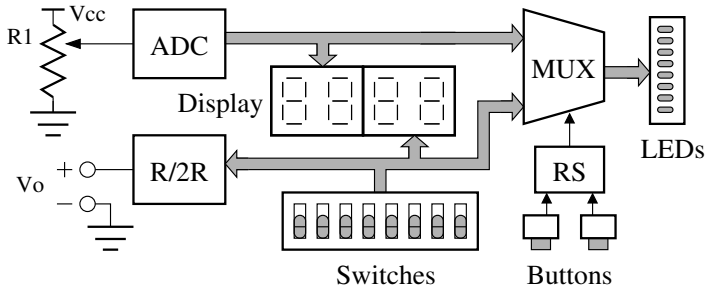

Figure 1: Conversion test circuit

Given an input within the range $V_{\min}$ to $V_{\max}$ and producing a n_b binary value, (1) gives the voltage discretization size δ_v and given an input, (2) gives the corresponding numerical code value. The $\lfloor\ \rfloor$ symbols here mean that the result is truncated, keeping the integer part. With a 3.3 Volt power supply and 8 bits, the discretization size δ_v is 12.891 mV. The ADC produce values \$00 to \$FF where the '\$' symbol indicates hexadecimal.

$$\delta_v = \frac{V_{\max} - V_{\min}}{2^{n_b}} \tag{1}$$

$$\text{code} = \left\lfloor \frac{V_i - V_{\min}}{\delta_v} \right\rfloor \tag{2}$$

The totem pole output type driver typically used in CMOS technology makes use the R-2R ladder network appealing as a DAC. The IRC application note on R-2R ladders[5] outlines the

general principle and application of R-2R ladder networks. Figure 2 is the ladder circuit constructed in the adapter board. The circuit can be understood by repeatedly applying Thevenin's well known theorem, starting at the left. Overall (3) is the Thevenin equivalent resistance and (4) is the Thevenin equivalent voltage.

$$R_{\text{thev}} = 1K \tag{3}$$

$$V_{\text{thev}} = \sum_{n=0}^{7} \frac{Vcc}{2^{8-n}} B_n \tag{4}$$

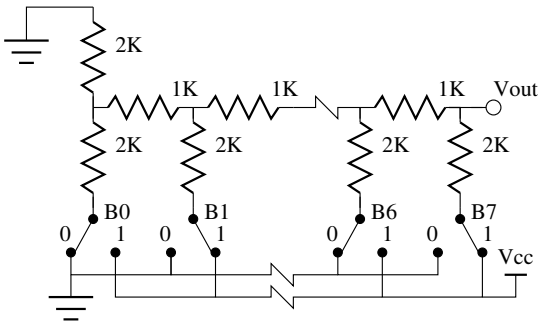

Figure 2: R-2R ladder circuit DAC

Figure 3 is the toolkit and a voltmeter. The configuration allows students to experiment with these basic conversion components. The white cable with label 2.8V is for JTAG, used for configuring the FPGA. The white region in the smaller upper board is a solder-less breadboard used here for the potentiometer. The LEDs to the left of the seven segment display is currently displaying the magnitude of the analog input.

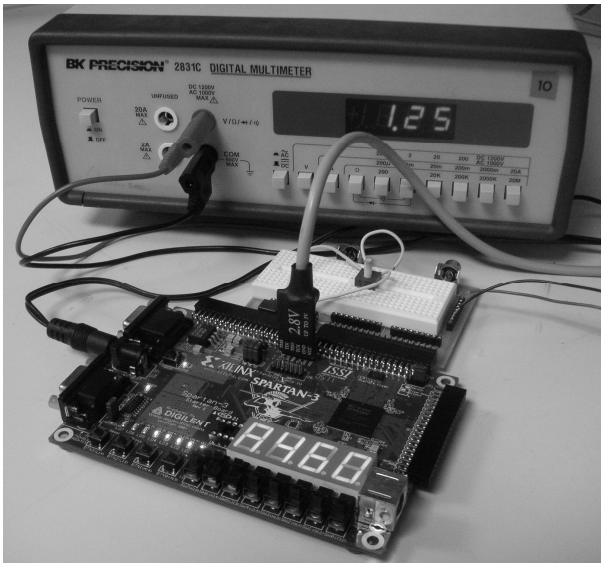

Figure 3: Toolkit and voltmeter

Sampling Experience

With the basics of conversions between analog and digital understood, we consider sampling. As in Figure 4, in a DTSP system a mechanism converts an analog input Vi to digital format, represented in the shaded region. Digital processing is performed numerically. Another mechanism converts the digital signal back to the analog realm, producing the output Vo. For now there is no processing. The goal is to investigate the notion of periodic sampling.

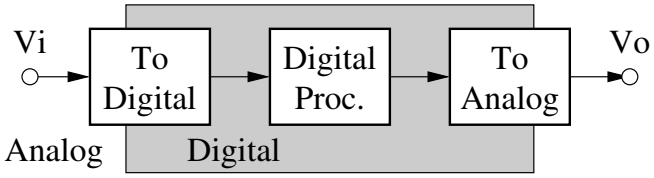

Figure 4: DTSP high level system view

The circuit in Figure 5 is used for input. In this experience we are not considering use of an anti-alias filter as one of point of this experience is to clearly demonstrate the effect caused by under sampling[8]. An important observation is that the ADC produces unsigned values. It is desirable for perturbations above the operating point to produce positive codes and below to produce negative codes. With the input at half the power supply voltage, the code produced is $80. To convert the unsigned ADC code to two's complement form we subtract this offset, but for this unique case it is enough to invert the most significant bit in the ADC code value.

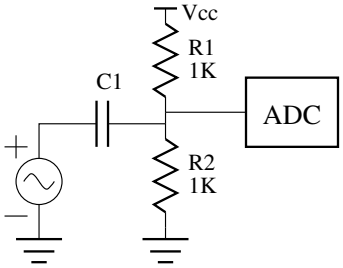

Figure 5: Converting analog to digital

Figure 6 is the corresponding schematic for logic implemented in the FPGA. The xdin box is a junction for signals going to and coming from the ADC. As discussed above, the xdin box inverts the most significant bit of the ADC value. The sdiv box produces the sample clock signal sclk. The xreg8 box is a register that inverts the most significant input bit and stores the value being converted back to analog.

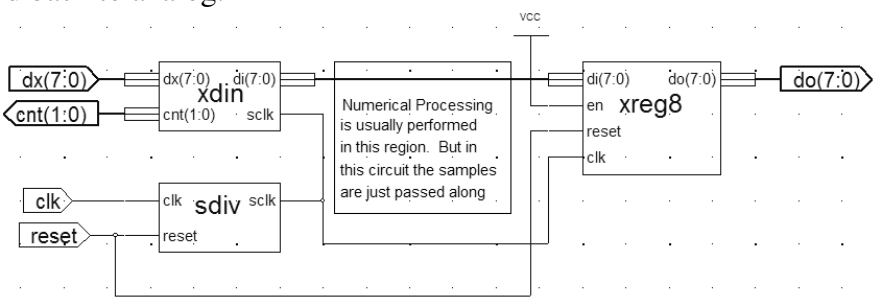

Figure 6: Sampling schematic for FPGA device

The output from the R-2R ladder is stepped as shown in Figure 8. To smooth the output a reconstruction filter is used. The Sallen and Key filter in Figure 7 is similar to that described by Huelsman and Allen[6] serves as a useful building block. The filter is second order and Table 1 lists three capacitor values and the corresponding cutoff frequencies. As further described by Huelsman and Allen, an additional resistor and capacitor can be used to increase the filter order to three. Here, the voltage divider formed with the DAC and R1 balances the overall DC gain to be unity. Figure 9 demonstrates aliasing. With 100 kHz sampling and a 95 kHz sine wave input, the output is 5 kHz.

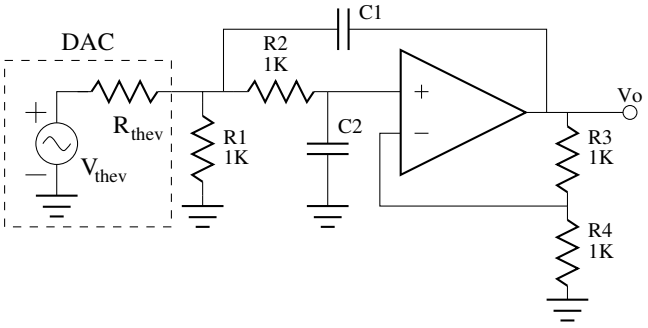

Figure 7: DAC and reconstruction filter

Table 1: Cutoff Frequencies

C1, C2	Fc
2.2nF	10.2 kHz
4.7nF	47.9 kHz
22nF	102 kHz

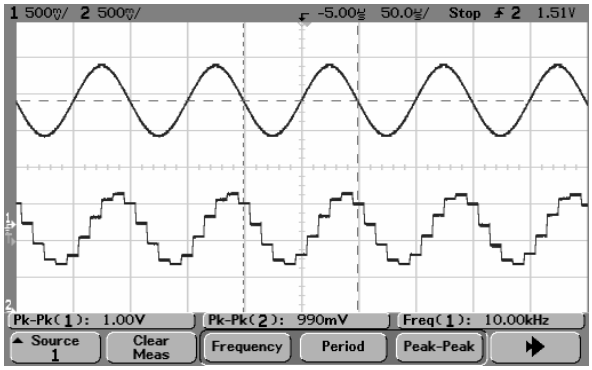

Figure 8: Without reconstruction filter

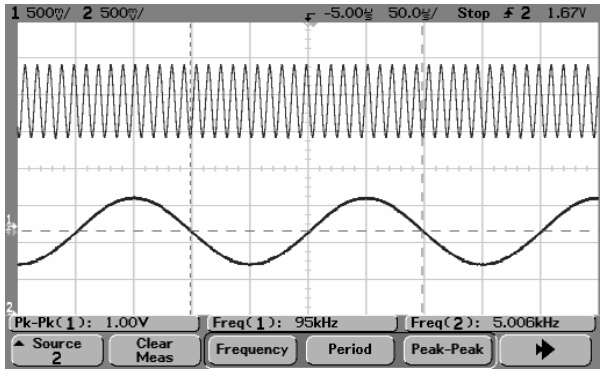

Figure 9: Demonstrating aliasing

Representing Signals as Fixed-Point Numbers

Using the toolkit involves getting used to the idea of fixed point numbers along with two's complement notation. The IEEE math libraries make it relatively easy to implement filters using an FPGA. The libraries include all the necessary fixed point math operators, namely addition, subtraction, and multiplication. As indicated by Xilinx[7] the Spartan-3 FPGA series has a number of features to support arithmetic. Carry logic and dedicated carry routing provides fast addition and subtraction. Multiplication can be performed in the FPGA fabric or with dedicated multipliers[7]. All these features make the use of fixed point numbers and two's complement notation particularly appealing.

The primary difference between floating point and fixed point numbers is how the radix point is positioned. With floating point numbers the radix point is dynamically positioned as each calculation is performed. The obvious cost of floating point numbers is increased circuit complexity and a potential reduction in speed. One cost with fixed point numbers is that the

designer decides in advance the radix point position for each signal. We use a simple 'aBc' notation that indicates 'a' bits and that the radix point is to the right of bit position 'c', where it is understood that the right-most bit position is number zero.

Precision and round-off are related issues. Floating point arithmetic handles such details silently with varying precision, based on the magnitude of the numbers involved. With fixed point numbers however the significance of each bit is known in advance as the designer must consciously address such details and decide when to drop bits. To avoid numerical overflow extra bits can be allocated as needed. Another situation arises in using a fixed discretization size, as small signals are not as well represented as larger ones. Ideally the filter response will be independent of amplitude. In some cases a poorly designed filter will display anomalies with small signals. As before, such problems can be eased by inserting more bits.

Selecting the number of bits and radix point position for constants is referred to as sizing the constants. As such, the position of the first non-zero bit after the sign bit is significant. Ideally the bit position is selected so that this bit is one. Equation (5) converts an aBc code value to its corresponding represented value.

$$\text{value} = \text{code}\, 2^{-c} \tag{5}$$

Math operations with fixed point numbers can have some interesting results. While it is true that the sum of two 'a' bit integers will fit into an 'a+1' integer, in adding eight such 'a' bit integers only three additional bits should be required. This paper arbitrarily sidesteps such issues by making the designer responsible for correctly sizing signals and constants. Table 2 lists the sizing of simple operators. Note that multiplication shifts the radix point. In performing these operations the designer must ensure that the sum, difference, or product will fit in the signal Y.

Table 2: Sizing of simple operators

Operation	Name	Sizing
W + X = Y	add	W, X. Y are 'aBc'
W – X = Y	subtract	W, X, Y are 'aBc'
W * X = Y	multiply	W is 'aBc', X is 'dBe' and Y is 'a+dBc+e'

IIR Filter Experience

IIR type filters tend to require modest resources. In mapping the components in such a filter block diagram directly to hardware it is possible produce one output sample for each clock cycle. Here we make use of the discussion of fixed point numbers to present a first order low-pass filter with a cutoff at 1 kHz. To design the filter I start with the continuous time model in Figure 10 and use the Matlab script lowpass1.m which applies the bilinear transform along with pre-warping to produce the coefficients for the corresponding discrete time system.

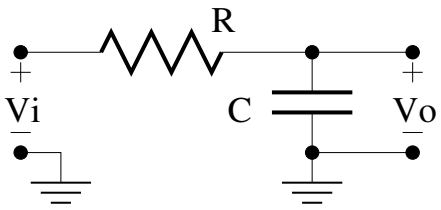

Figure 10: Low-pass RC filter

```
% lowpass1.m produce filter coefficients
% using Bilinear transform and prewarp.

% Design Parameters
Fc = 1e3    % filter center frequency
Fs = 100e3  % sampling frequency
Wc = 2*pi*Fc;

% Model the continuous time system function
Num = [0 1];
Den = [1/Wc 1];

% Transform and prewarp to match cutoff
[N,D] = BILINEAR(Num,Den,Fs,Fc)
```

Figure 11 is the corresponding DTSP filter model. The sizing of the coefficients K1, K2, and K3 is summarized in Table 3, showing the value represented. Note that despite that few bits are used, the percent difference error with the theoretical value is within two tenths of a percent.

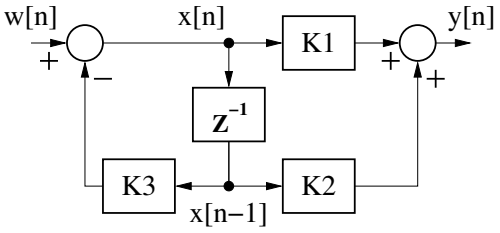

Figure 11: DTSP filter model

$$N = [\mathrm{K1}\ \mathrm{K2}] = [0.0304687\ \ 0.0304687] \tag{6}$$

$$D = [1\ -\mathrm{K3}] = [1\ \ -0.939063] \tag{7}$$

$$H(z) = \frac{y(z)}{w(z)} = \frac{\mathrm{K1}\,z + \mathrm{K2}}{z + \mathrm{K3}} \tag{8}$$

Table 3: Fixed point coefficient approximations

Coefficient	Size	Code	Represents	% Diff. Error
K1	5B4	15	0.937500	– 0.168%
K2	8B12	125	0.0305176	+ 0.160%
K3	8B12	125	0.0305176	+ 0.160%

Great care must be exercised in sizing internal signals. To obtain speed with an FPGA it is necessary to use as few bits as possible to represent each signal. With IIR filters, the internal signals tend to experience a magnification effect, which is analogous to the storage of energy in a continuous time filter component like a capacitor or inductor. For this low-pass filter the effect is seen in the step response. Given a step input value w_s the steady state value for x can be approximated by recognizing the familiar geometric power series.

$$x_s = w_s \sum_{n=0}^{\infty} \mathrm{K3}^n = \frac{w_s}{1+\mathrm{K3}} = 16\,w_s \tag{9}$$

Unlike floating point numbers where the details involving precision are handled by a processor in silent fashion, with fixed point numbers the designer is responsible. To assist in this regard, Figure 12 is the so-called plumbing diagram for the example first order low-pass DTSP filter. The figure is produced from Figure 11 by inserting slash boxes as necessary to resize signals and inserting sizing information. Here, the designer chose to resize the input to include four extra

bits to the left side for the magnification effect and eight extra bits to the right to increase precision. An extra bit is inserted into the *x* and delayed *x* value to avoid overflow in addition. Following the product with K3, four bits are dropped from the right of *x*. With K1 and K2 equal, multiplication with K1 is moved after the addition bubble. In producing the output the final slash box drops 21 bits.

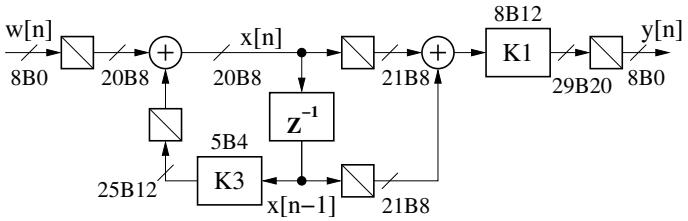

Figure 12: Plumbing diagram for low pass filter

Figure 13 is the corresponding schematic drawn with the Xilinx ISE capture tools. Each symbol is drawn and corresponds to a simple VHDL module. The ext boxes resize by adding bits. The multcut boxes multiply by a constant value and cut bits. The reg20 box introduces a delay. The add20x box first inserts a bit to each value before adding.

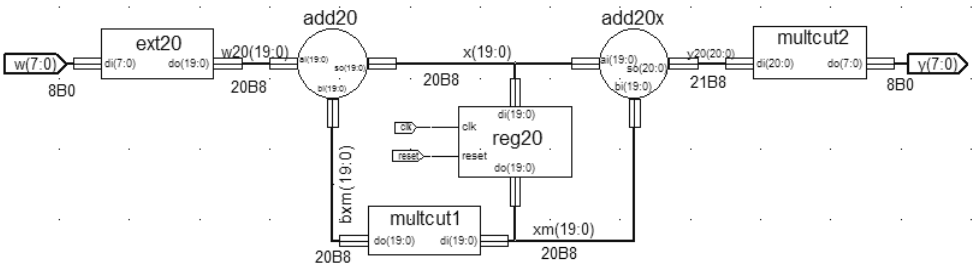

Figure 13: Schematic for low pass filter

Figure 14 is the input and output for the DTSP low-pass filter, operating near the 3dB frequency. With a 1 Volt pk-pk input at the 3dB frequency of 1 kHz the output should be 0.707 Volts. Here, near the 3 dB frequency the output is 0.700 Volts pk-pk.

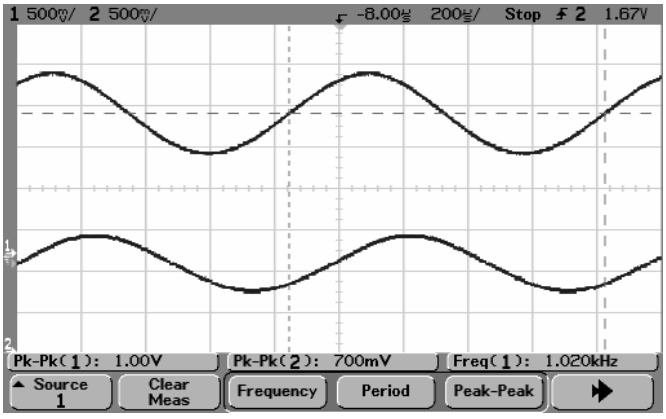

Figure 14: Low pass filter at 3dB frequency

FIR Filter Architecture

The primary issue with implementing a FIR filter directly in hardware is the sheer size of the circuitry involved. Multipliers implemented directly in FPGA fabric require significant resources and are slow. Dedicated hardware multipliers are fast, but are modest in number, varying by device model. The commonly used XC3S200 device used here has 12 dedicated hardware multipliers which is limiting. In contrast the XC3S5000 has 104 dedicated multipliers. Given the speed of hardware multipliers, for respectable sample rates we can use time multiplexing. Here we consider one such FIR filter.

The following presentation is intentionally brief. Many DSP text books, including Oppenheim and Schafer[8] as well as Ingle and Proakis[9] provide useful background. As in (10), when the input x is a unit impulse, the output y is the corresponding impulse response. It is reasonable to assume for such circuits that the impulse response is causal or zero for n less than 0.

$$\begin{aligned} x[n] &= \delta[n] \\ y[n] &= h[n] \end{aligned} \tag{10}$$

To produce the convolution sum we assume the system is linear and time invariant. If the input is a scaled and delayed impulse, the output will be scaled and delayed in the same fashion.

$$\begin{aligned} x[n] &= a[k]\,\delta[n-k] \\ y[n] &= a[k]\,h[n-k] \end{aligned} \tag{11}$$

Again, applying linearity, given an input x that is a sum of scaled and delayed impulses, the output will also be sum of scaled and delayed impulse responses.

$$y[n] = \sum_{k=0}^{n} a[k]\,h[n-k] \tag{12}$$

Assume the impulse response $h[n]$ is zero for n larger than m. We split the sigma operator to appear s times. At this point, taking the convolution sum further is only practice in notation.

$$y[n] = \sum_{k_{s-1}=n_{s-1}}^{n} a[k_{s-1}]\,h[n-k_{s-1}] + \Lambda + \sum_{k_2=n_1}^{n_2-1} a[k_2]\,h[n-k_2] + \sum_{k_1=n-\lambda}^{n_1-1} a[k_1]\,h[n-k_1] \tag{13}$$

where $n-\lambda < n_1 < n_2 < \mathrm{K} < n_{s-1} < n$

To implement each sum in (13) it is convenient to store each part of the impulse response in a read-only memory or ROM and the input values in RAM, having one circuit like that in Figure 15 for each of the smaller sum terms. The RAM is asynchronous-read, synchronous-write, which means a write is committed to memory at a rising clock edge. An accumulate-and-dump circuit (ADM) is produced with a register that clears in synchronous fashion along with a parallel load register. The parallel load register obtains the final sum as the first register clears in preparation for accumulating the next sum. Each such stage forms the sum of M products.

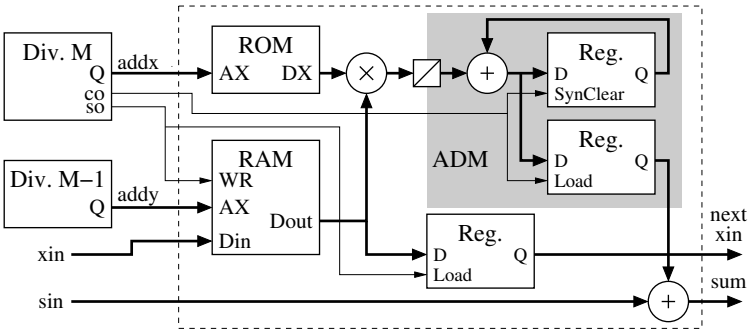

Figure 15: Partial convolution circuit

To understand Figure 15, consider the timing diagram in Figure 16. In a manner like the way the convolution sum flips one signal time-wise, here the upper counter decrements while the lower increments. The ADM loads the final sum when the upper counter signal co is asserted, when the upper counter value is zero. In the next cycle WR is asserted. This reads the oldest sample at the same time the newest sample is being written to RAM. The next time co is asserted addy is increased by one, compared to the previous time.

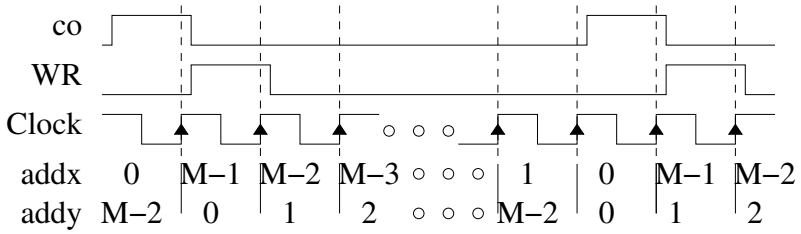

Figure 16: Convolution timing

Note that Figure 15 and Figure 16 do not outline how reset is applied. The RAM has no direct mechanism for reset. During reset, the data input is gated for M clock cycles so that zeros are written to RAM. During this same time, because of the reset, the ADM also produces zero. To summarize, the convolution sum provides an effective FIR filter architecture. A primary benefit is the way the architecture allows performance to be traded off for complexity. With s convolution stages, each storing M samples in ROM, the result is a FIR filter with $M \cdot s$ taps.

FIR Filter Experience

To provide a simple example that is also fairly self explanatory, I wrote a script for Matlab or Octave to produce samples of an exponentially decaying 10 kHz sine wave signal. Such an impulse response corresponds to a resonance. Such a circuit behaves as a band pass filter. The time constant is 20 cycle or 0.2 milliseconds and samples are taken for three time periods or 60 samples. Figure 17 is the corresponding schematic produced using the Xilinx ISE tools.

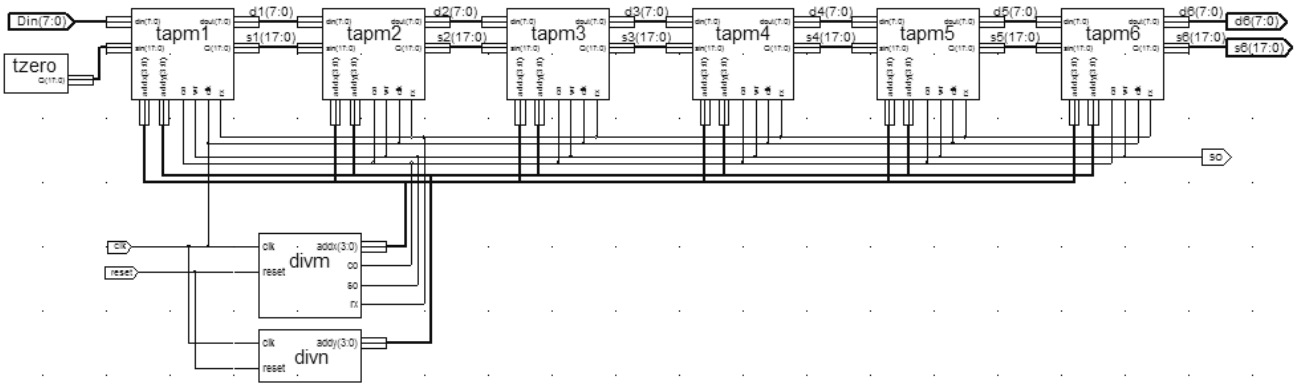

Figure 17: FIR filter constructed with six convolution stages

To provide more detail, Figure 18 is a closer view of the first few stages along with the counter logic. The design can be simplified, combining similar signals into larger busses. For more detail regarding each signal, please refer back to Figure 15.

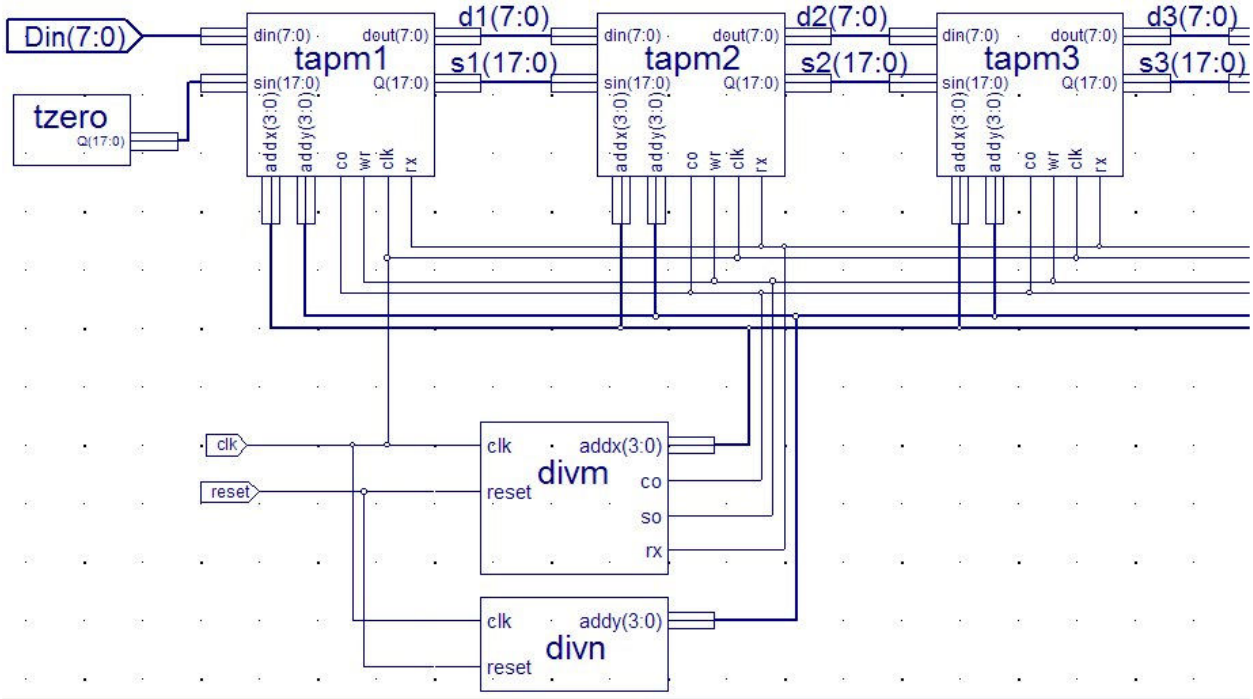

Figure 18: First few stages of FIR filter

Figure 19 is the corresponding filter output at the higher 3dB frequency. In the case of this filter, one sample is produced every ten clock cycles. With a 100 kHz sample frequency the clock period is only 1 MHz. Use of the convolution circuit in this way allow for a trade-off between speed and circuit complexity.

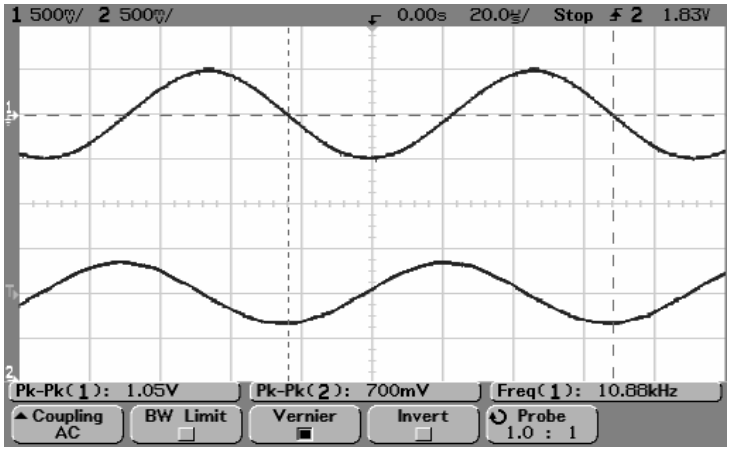

Figure 19: Bandpass filter 3dB point

Conclusion

It is our goal that the discrete time signal processing toolkit will be useful to students learning digital signal processing principles as well as to advanced students working on their own projects. The toolkit is being developed for use in a digital signal processing course. As an introductory tool, the toolkit allows a deductive approach where students investigate existing systems. Advanced students ready for a more inductive approach can use the toolkit in their own projects. The toolkit is multipurpose that along with course materials provides several methods to process signals. Future work will involve the use of a soft core digital signal processor. Details on the project are at the project website[1].

Bibliography

1. Project website, http://uhaweb.hartford.edu/jmhill/projects/DTSPkit/index.htm
2. Digilent, Inc., http://www.digilentinc.com/
3. Analog Devices, Inc., http://www.analog.com/
4. Xilinx, Inc., http://www.xilinx.com/
5. Jerry Seams, "R/2R Ladder Networks, Application Note AFD006," copyright 1998 by International Resistive Company, Inc. http://www.irctt.com/pdf_files/LADDERNETWORKS.pdf
6. L. P. Huelsman and P. E. Allen, Introduction to the Theory and Design of Active Filters, copyright 1980 by McGraw-Hill, Inc.
7. Xilinx, "Using Embedded Multipliers in Spartan-3 FPGAs," XAPP467, May 13, 2003, http://www.xilinx.com/bvdocs/appnotes/xapp467.pdf
8. Alan V. Oppenheim and Ronald W. Schafer, Digital Signal Processing, copyright 1975 by Alan V. Oppenheim and Bell Telephone Laboratories, Inc., published by Prentice-Hall, Inc.
9. Vinay K Ingle and John G. Proakis, Digital Signal Processing using Matlab, copyright 2000 by Brooks/Cole Publishing Company.